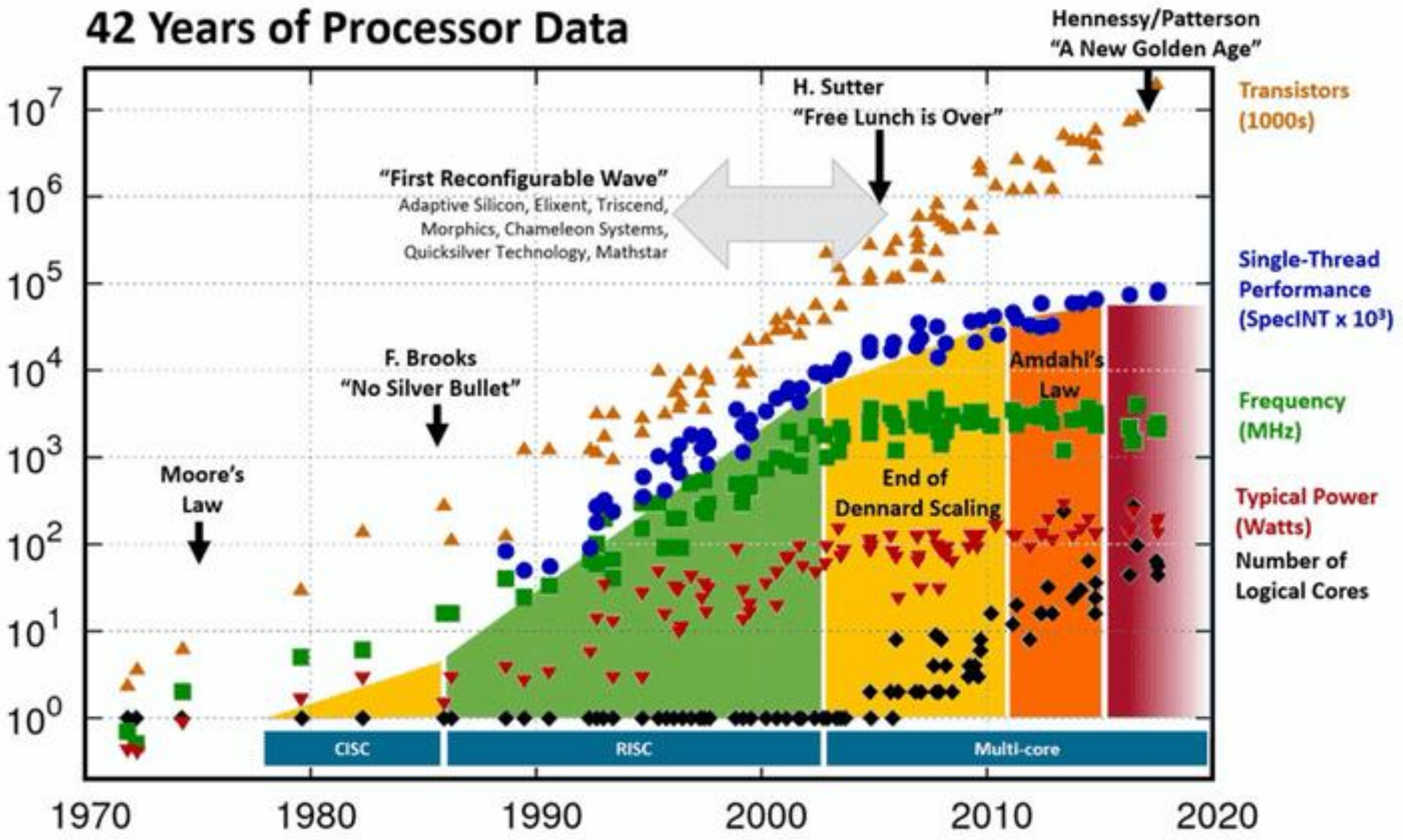
EVOLVING HPC APPLICATIONS FOR PERFORMANCE PORTABILITY – LESSONS LEARNT FROM OP-DSLS

Gihan Mudalige
Royal Society Industry Fellow
Reader in High Performance Computing
Department of Computer Science, University of Warwick
g.mudalige@warwick.ac.uk

Joint work with:
Istvan Reguly @ PPCU
Kamalavasan Kamalakannan, Arun Prabhakar, Archie Powell and others at the HPSC group @ Warwick
Neil Sandham and team @ Southampton, Dario Amirante @ Surrey
Mike Giles @ Oxford, Sylvain Laizet, Paul Kelly and many more @ Imperial College London
Rolls-Royce plc., NAG, UCL, STFC, IBM and many more.

UK Turbulence Consortium Annual Meeting March 2022

SINGLE THREAD SPEEDUP IS DEAD – MUST EXPLOIT PARALLELISM



Hennessy and Patterson, Turing Lecture 2018, overlaid over "42 Years of Processors Data"
https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/;"First Wave" added by Les Wilson, Frank Schirrmeister
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

The Hail Mary pass !

http://www.theemike.com/mikes-free-football-comic-book-hail-mary-pass/

"The semiconductor industry threw the equivalent of a Hail Mary pass when it switched from making microprocessors run faster to putting more of them on a chip - doing so without any clear notion of how such devices would in general be programmed."

David Patterson, University of California - Berkeley 2010

Diverse Hardware Landscape – Compounded by the Race to Exascale !

- **Traditional CPUs**
  - Intel, AMD, ARM, IBM
  - multi-core (> 20 currently)
  - Deep memory hierarchy (cache levels and RAM)
  - longer vector units (e.g. AVX-512)
- **GPUs**
  - NVIDIA (A100), AMD (MI200) , Intel (Xe GPUs)
  - Many-core (> 1024 simpler SIMT cores)
  - CUDA cores, Tensor cores
  - Cache, Shared memory, HBM (3D stacked DRAM)
- **Heterogeneous Processors**
  - Different core architectures over the past few years
  - ARM *big.LITTLE*
  - NVIDIA *Grace.Hopper*
- **XeonPhi (discontinued)**
  - Many-core – based on simpler x86 cores
  - MCDRAM (3D stacked DRAM)
- **FPGAs**
  - Xilinx (AMD) and Intel
  - Various configurations
  - Low-level language / HLS tools for programming
  - Significant energy savings
- **DSP Processors**
  - Matrix 2000+ (MTP) DSP accelerator
  - [Yet to be announced Chinese Exascale system ?]
- **TPUs (e.g. from Google), IPUs …**

… Custom ASICs driven by AI ... in the cloud.

- **Domain specific Hardware …**
- **Quantum [?]**


But .. Even more diverse ways to programming them !

OpenMP, SIMD, CUDA, OpenCL, OpenMP4.0, OpenACC, SYCL/OneAPI, HIP/ROCm, MPI, PGAS, Task-based (e.g Legion) ….

- ❑ Open standards (e.g OpenMP, SYCL)
  - So far have not been agile to catch up with changing architectures
- ❑ Proprietary models (e.g. CUDA, OpenACC, ROCm, OneAPI)
  - Restricted to narrow vendor specific hardware
- ❑ Need different code-paths/parallelization schemes to get the best performance
  - E.g. Coloring vs atomics vs SIMD vs MPI vs Cache-blocking tiling for unstructured mesh class of applications
- ❑ What about legacy codes ? There is a lot of FORTRAN code out there !

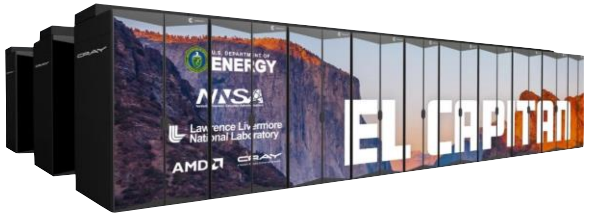
❑ What would an Exa-scale machine architecturally look like ?

- Perlmutter - Over 100 PFLOP/s - AMD EPYC CPUs (Milan) with NVIDIA A100 GPUs
- Aurora - 1 EFLOP Intel Xeon CPUs (Sapphire Rapids) with Intel Xe GPUs
- Frontier - 1.5 EFLOP/s AMD EPYC CPUs (Milan) with AMD Instinct GPUs
- El Capitan - 2 EFLOP/s AMD EPYC CPUs (Genoa) with AMD Instinct GPUs

---

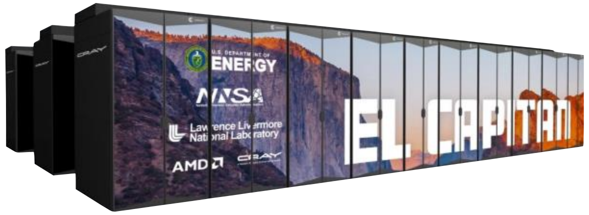
- LUMI - 0.5 EFLOP/s AMD EPYC CPUs with AMD Instinct GPUs
- LEONARDO - 0.3 EFLOP/s - Intel Xeon CPUs (Sapphire Rapids) with NVIDIA A100 GPUs
- MareNostrum5 - 2 distinct 100+ PFLOP/s systems possibly based on ARM/RISC-V

- ARCHER2- 28 PFLOP/s AMD EPYC CPUs (Rome)
- Many Tier-2 systems in the UK - Isambard-2 – ARM A64FX | Baskerville - NVIDIA A100 GPUs



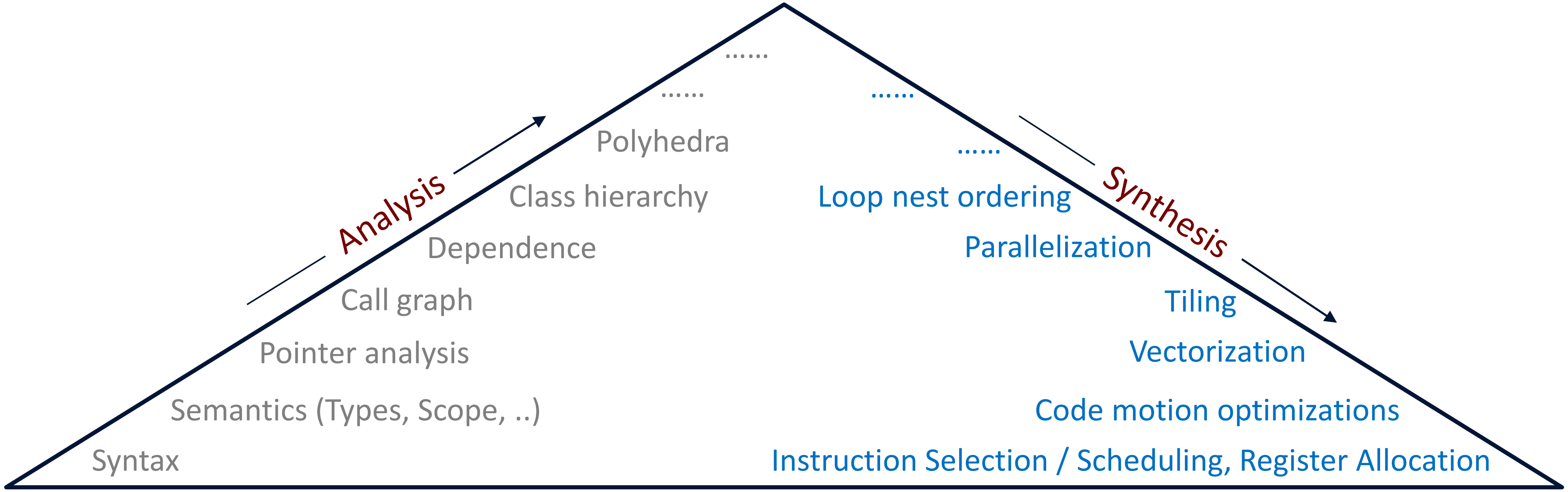
Software Challenge – A moving target

- Each new platform requires new performance tuning effort
  - Deeper memory/cache hierarchies and/or shared-memory (including non-coherent)
  - Multiple (heterogeneous) memory spaces (device memory/host memory/near-chip memory)
  - Complex programming skills set needed to extract best performance on the newest architectures
- Not clear which architectural approach is likely to win in the long-term
  - Cannot be re-coding applications for each new type of architecture or parallel system
  - Nearly impossible for re-writing legacy codes
- Need to future-proof applications for their continued performance and portability
  - If not – significant loss of investment
  - Applications will not be able to make use of emerging architectures



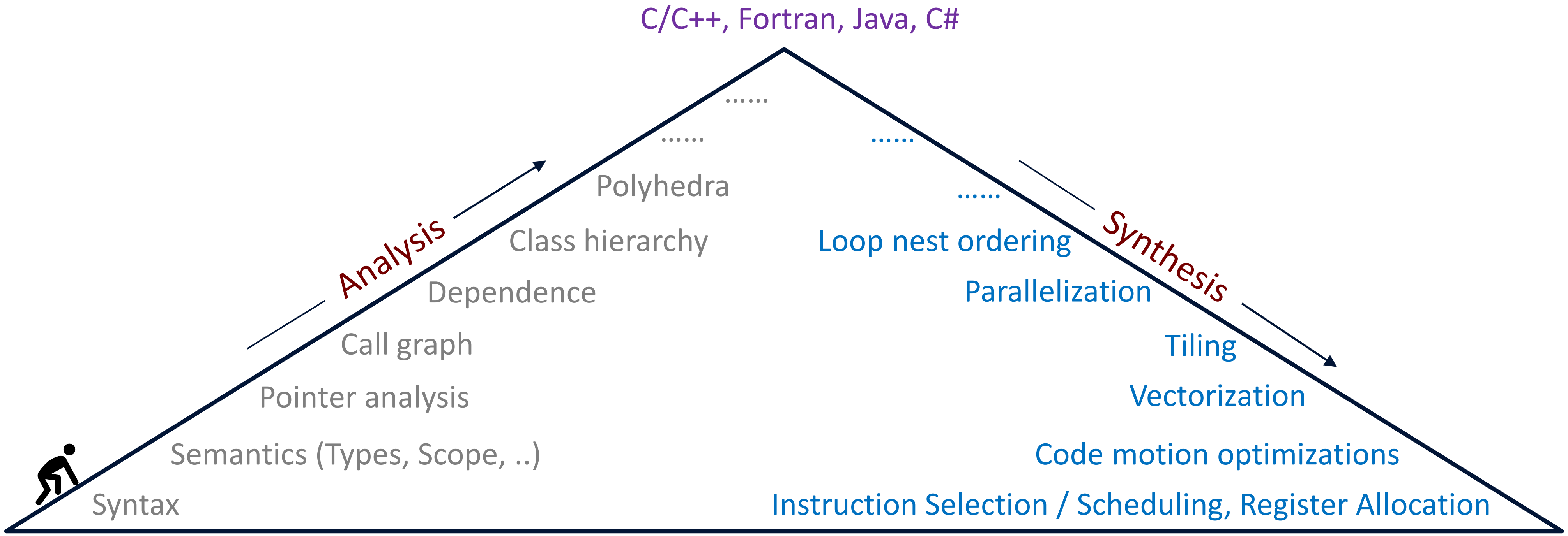
Outline

- ❑ Motivation ✓
- ❑ Raising the Level of Abstraction
- ❑ Oxford Parallel Libraries [OP-DSLs] – OP2 and OPS
- ❑ Evolving Production Codes – Hydra to OP2-Hydra
- ❑ Projects and Codes Using OP-DSLs
- ❑ Lessons Learnt
- ❑ Future Work
- ❑ Conclusions



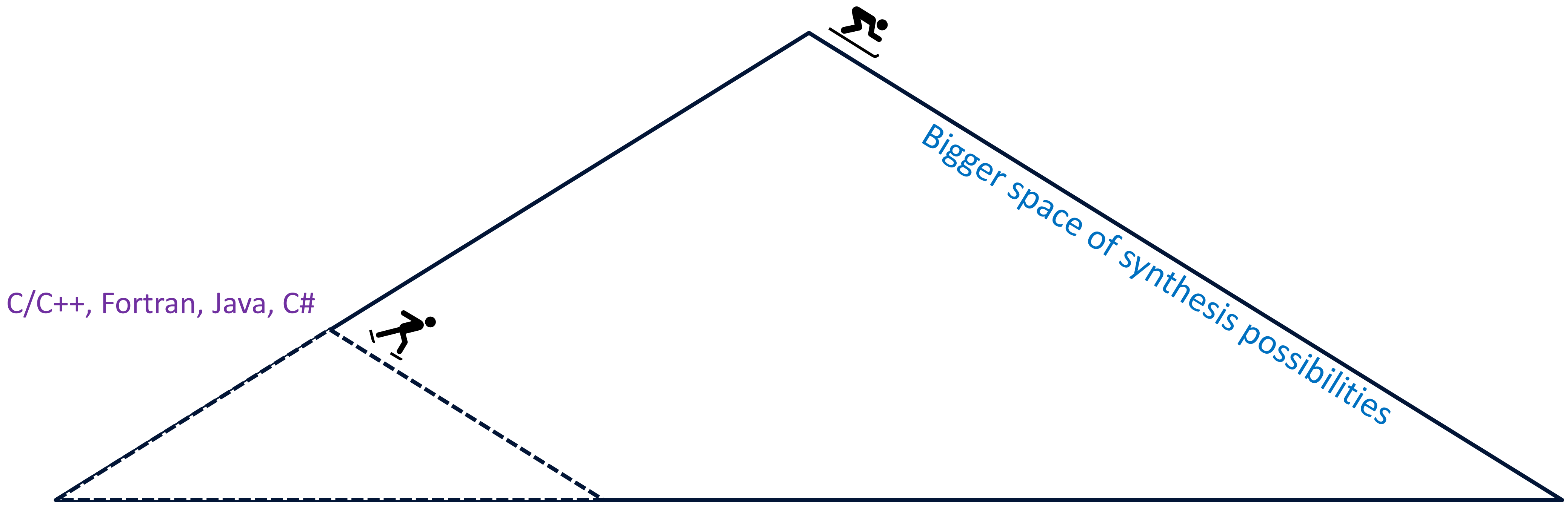
Adapted from: *Synthesis versus Analysis: What Do We Actually Gain from Domain-Specificity?* Keynote talk at the LCPC 2015. Paul H. J. Kelly (Imperial College London)

- Classical compiler has two halves
  - *Analysis* – gather information about the programme
  - *Synthesis* – generate target code
- The higher you can get to in *analysis* the bigger the space for code synthesis possibilities



Adapted from: *Synthesis versus Analysis: What Do We Actually Gain from Domain-Specificity?* Keynote talk at the LCPC 2015. Paul H. J. Kelly (Imperial College London)

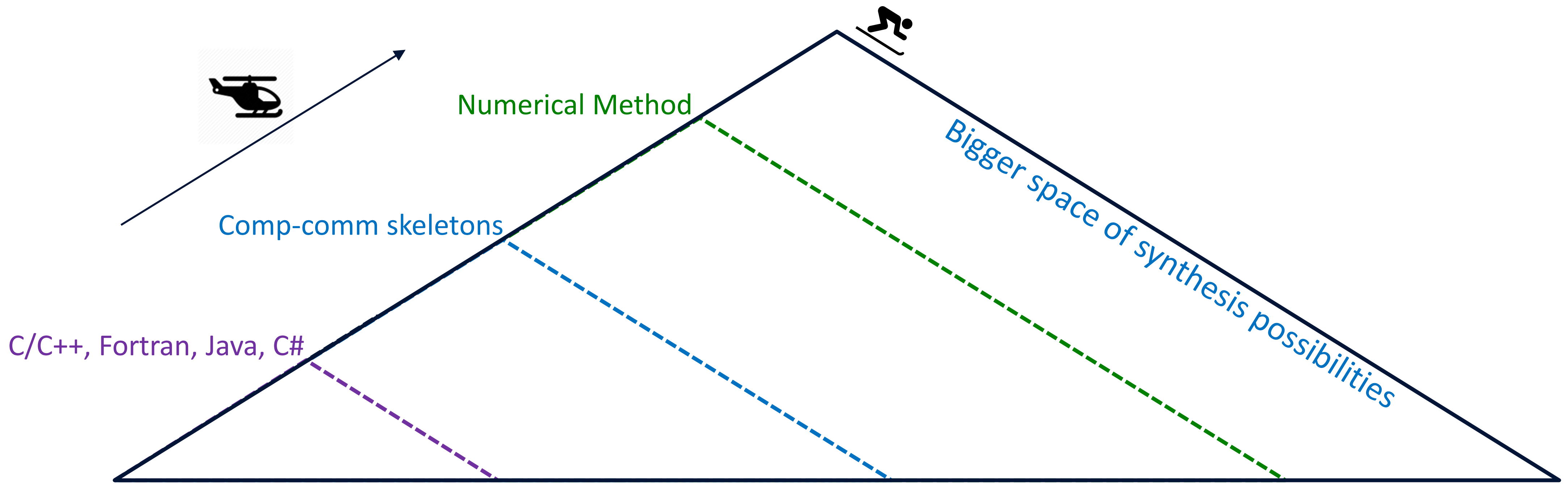
- If you start at a lower level – climbing higher is a struggle
  - Difficult to ensure optimizations are safe (e.g. data races, pointer aliasing)
  - Sometimes, impossible to extract richer information (e.g. data partitioning/layouts, memory spaces)
  - Limits the optimizations possible
- Compounding the issue - the way code is written by (most) people will not be easy to analyze !



Adapted from: *Synthesis versus Analysis: What Do We Actually Gain from Domain-Specificity?* Keynote talk at the LCPC 2015. Paul H. J. Kelly (Imperial College London)

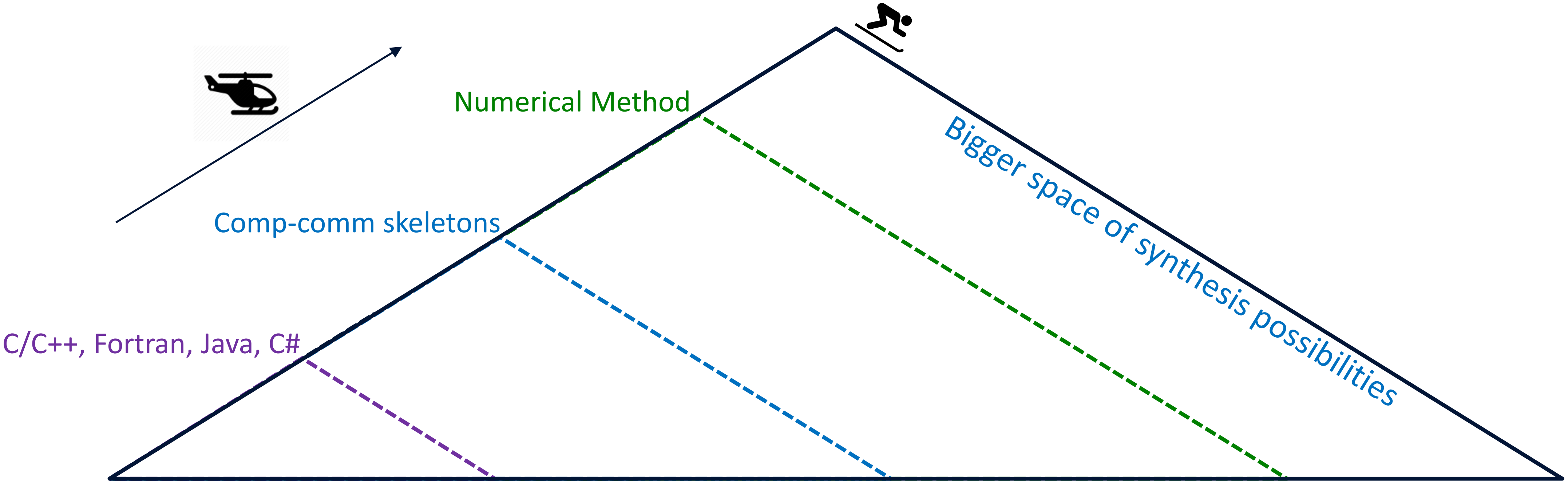
- If you can start higher
  - Results in a bigger space of code synthesis possibilities
  - Could they give the same (or better) performance as code written by hand ?
  - Could these possibilities include targeting different (parallel) architectures ?

Domain Specific Abstractions



Adapted from: *Synthesis versus Analysis: What Do We Actually Gain from Domain-Specificity?* Keynote talk at the LCPC 2015. Paul H. J. Kelly (Imperial College London)

- Rise the abstraction to a specific domain of variability
- Concentrate on a narrower range (class) of computations
  - Computation-Communications skeletons - Structured-mesh, Unstructured-mesh, … 7 Dwarfs [Colella 2004] ?
  - (higher) Numerical Method - PDEs, FFTs, Monte Carlo …
  - (even higher) Specify application requirements, leaving implementation to select radically different solution approaches



Adapted from: *Synthesis versus Analysis: What Do We Actually Gain from Domain-Specificity?* Keynote talk at the LCPC 2015. Paul H. J. Kelly (Imperial College London)

- If you get the abstraction right, then:
  - Can isolate numerical methods from mapping to hardware
  - Can reuse a body of optimizations/code generation expertise/techniques for this class (or numerical method) to match target hardware

How Do We Raise The Level of Abstraction ?

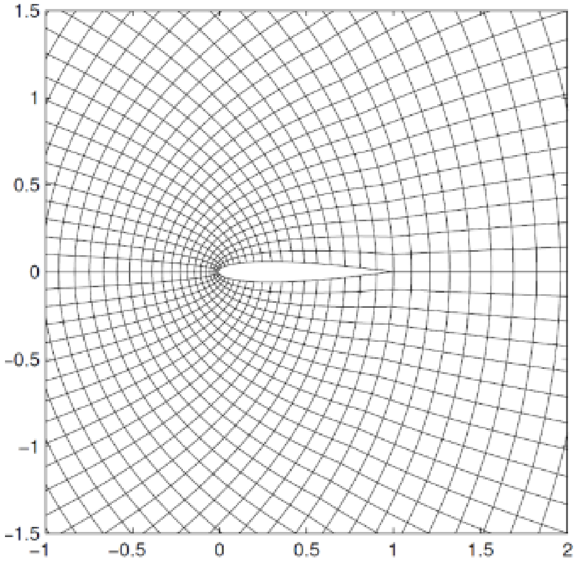
- Domain Specific API
  - Get application scientists to pose the solution using domain specific constructs – provided by the API
  - Handling data done only using API – contract with the user

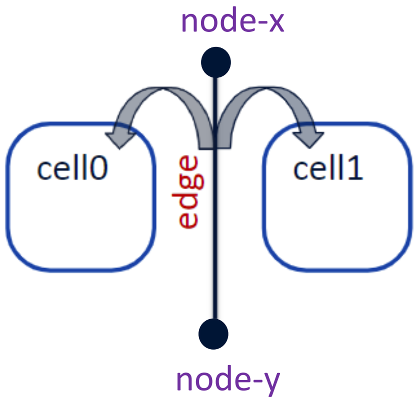
- Restrict writing code that is difficult (for the compiler) to reason about and optimize
  - *"OP2 and OPS are a straightjacket"* – Mike Giles

- Implementation of the API left to a lower level
  - Target implementation to hardware - automatically generate implementation from specification for the context
  - Generate code in best parallelization model – open standards or proprietary !
  - We know how to best optimize to that specific hardware – reuse these best optimizations
  - Exploit domain knowledge for better optimisations

OP2 – Unstructured-mesh Applications Domain

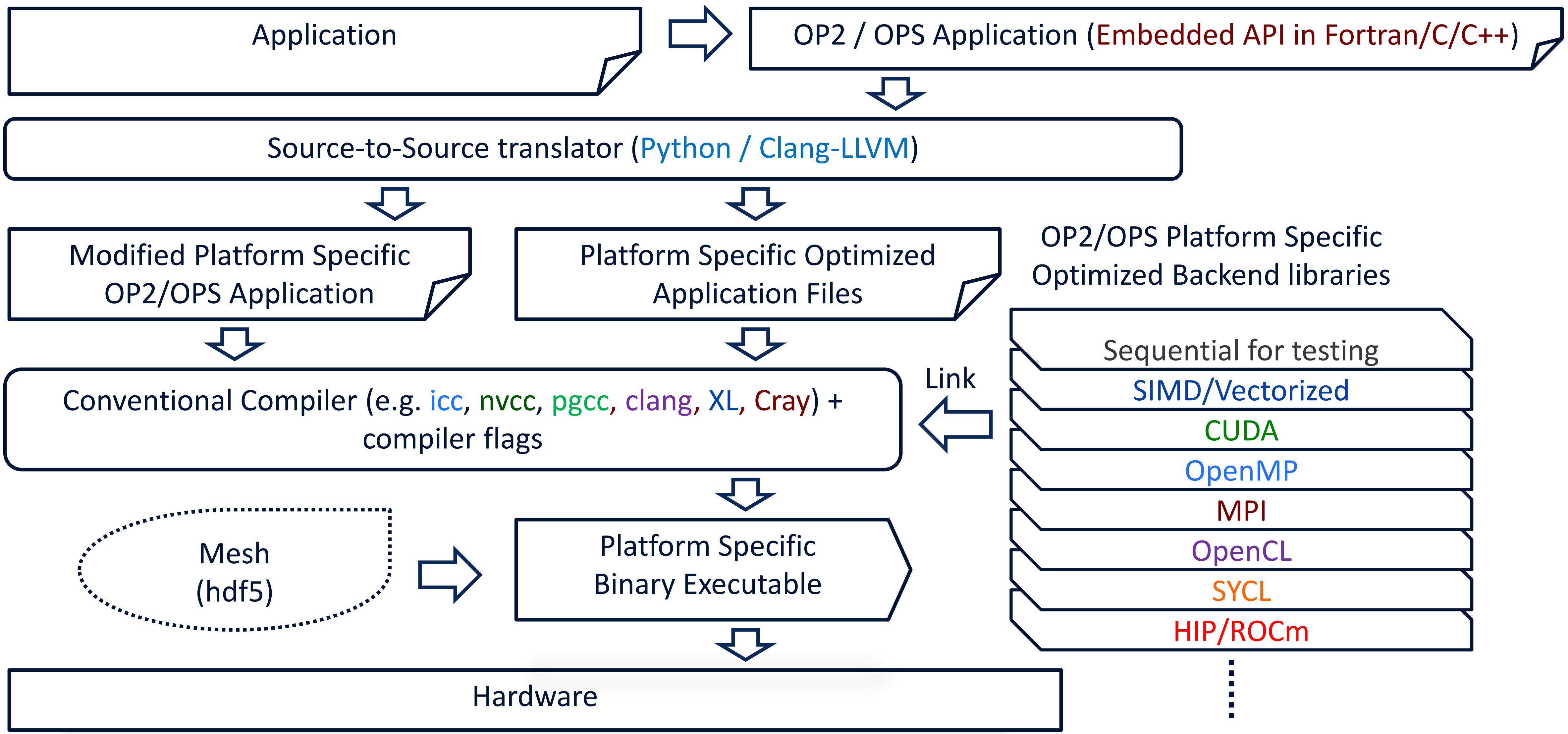
```fortran
! Declaring the mesh with OP2
! sets
call op_decl_set(nnode,nodes,'nodes')
call op_decl_set(nedge,edges,'edges')
call op_decl_set(ncell,cells,'cells')
! maps
call op_decl_map(edges,nodes,2,edge ,pedge ,'pedge' )
call op_decl_map(edges,cells,2,ecell,pecell,'pecell')
! data
call op_decl_dat(nodes,2,'real(8)',x,p_x,'p_x')
call op_decl_dat(cells,4,'real(8)',q,p_q,'p_q')
call op_decl_dat(cells,1,'real(8)',adt,p_adt,'p_adt')
call op_decl_dat(cells,4,'real(8)',res,p_res,'p_res')
```

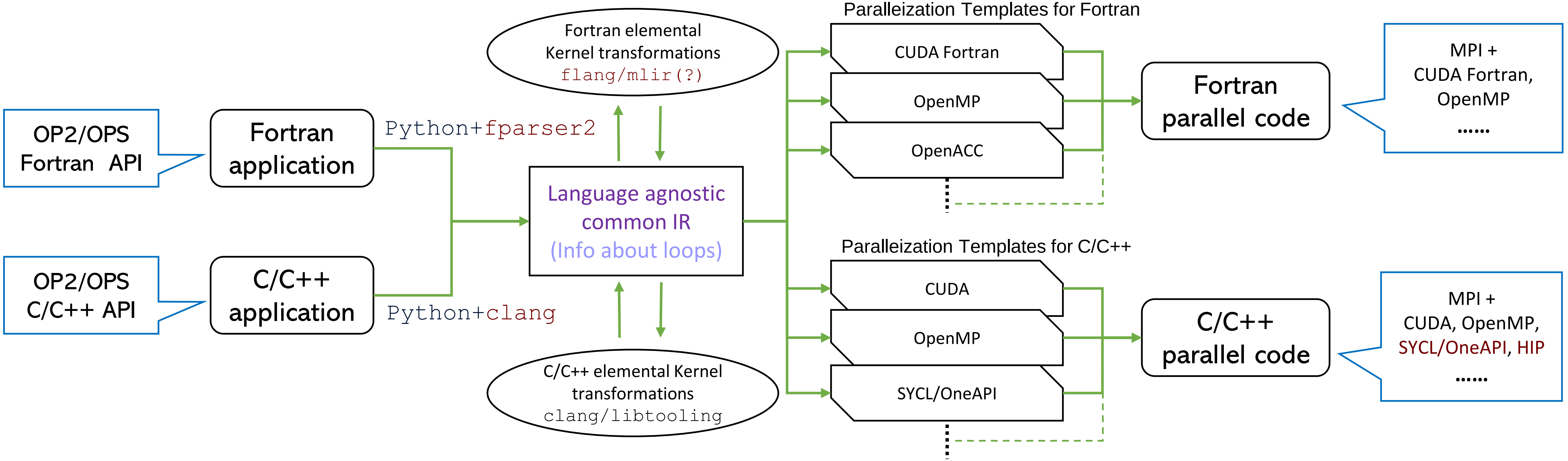





```fortran
! Elemental kernel
subroutine res_calc(x1,x2,q1,q2,adt1,adt2,res1,res2)
  IMPLICIT NONE
  REAL(kind=8), DIMENSION(2), INTENT(IN) :: x1
  REAL(kind=8), DIMENSION(2), INTENT(IN) :: x2
  ...
  REAL(kind=8) :: dx,dy,mu,ri,p1,vol1,p2,vol2,f
  dx = x1(1) - x2(1)
  dy = x1(2) - x2(2)
  ...
  f = 0.5 * (vol1 * q1(1) + vol2 * q2(1)) + &
  &     mu * (q1(1) - q2(1))
  res1(1) = res1(1) + f
  res2(1) = res2(1) - f
  ...
! Calculate flux residual - parallel loop over edges
call op_par_loop_8 (res_calc, edges, &
& op_arg_dat(x,   1, edge,  2,"real(8)", OP_READ), &
& op_arg_dat(x,   2, edge,  2,"real(8)", OP_READ), &
& op_arg_dat(q,   1, ecell, 4,"real(8)", OP_READ), &
& op_arg_dat(q,   2, ecell, 4,"real(8)", OP_READ), &
& op_arg_dat(adt, 1, ecell, 1,"real(8)", OP_READ), &
& op_arg_dat(adt, 2, ecell, 1,"real(8)", OP_READ), &
& op_arg_dat(res, 1, ecell, 4,"real(8)", OP_INC ), &
& op_arg_dat(res, 2, ecell, 4,"real(8)", OP_INC ))
```

OP2 – Application development workflow





- Simplest code generation / translation
  - Intermediate representation is simply the loop descriptions + elemental kernels
  - Generated parallel code can be viewed and understood by a human !

- Multi-layered – no opaque / black box layers
- Built with well supported / long-term technologies - Python, Clang/libtooling, [flang?, mlir?]

OP2 – Generated Code - CPU

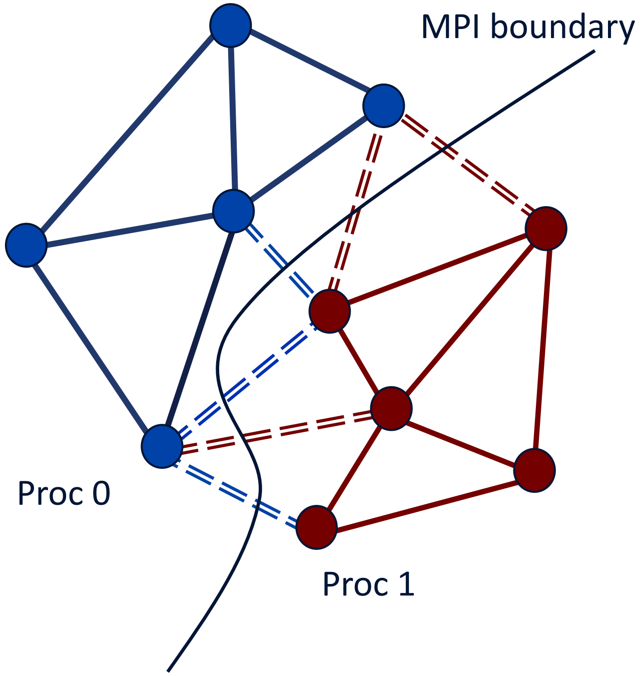
```fortran
! elemental kernel
SUBROUTINE res_calc(x1,x2,q1,q2,adt1,adt2,res1,res2)
...
END SUBROUTINE

! wrapper function - calls elemental kernel
SUBROUTINE op_wrap_res_calc( ... )
...
...
  DO i1 = bottom, top-1, 1
    IF (mod(i1,testfreq).eq.0) THEN
      call op_mpi_test_all(argc,args)
    END IF
    map1idx = opDat1Map(1 + i1 * opDat1MapDim + 0)+1
    map2idx = opDat1Map(1 + i1 * opDat1MapDim + 1)+1
    map3idx = opDat3Map(1 + i1 * opDat3MapDim + 0)+1
    map4idx = opDat3Map(1 + i1 * opDat3MapDim + 1)+1
    ! kernel call
    CALL res_calc(
         opDat1Local(1,map1idx), opDat1Local(1,map2idx),
         opDat3Local(1,map3idx), opDat3Local(1,map4idx),
         opDat5Local(1,map3idx), opDat5Local(1,map4idx),
         opDat7Local(1,map3idx), opDat7Local(1,map4idx))
    END DO
END SUBROUTINE
```

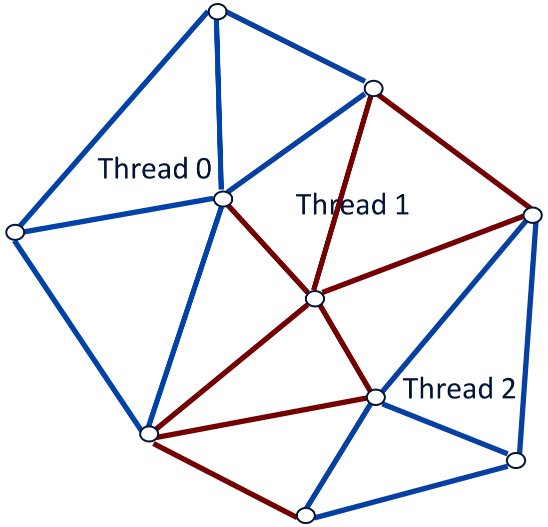
```fortran
! host function - setting up pointers and indirect accesses
SUBROUTINE res_calc_host( userSubroutine, set, opArg1, &
& opArg2, & opArg3, & opArg4, opArg5, opArg6, opArg7, opArg8)

  ! MPI halo exchanges
  n_upper = op_mpi_halo_exchanges(...)
  ...
  ...
  ! set up c to Fortran pointers
  CALL c_f_pointer(opArg1%data,opDat1Local, ...)
  CALL c_f_pointer(opArg1%map_data,opDat1Map, ...)
  ...
  ...
  ! compute over core iterations/elements
  CALL op_wrap_res_calc( opDat1Local, opDat3Local, &
                       & opDat5Local, opDat7Local, &
                       & opDat1Map, opDat1MapDim,  &
                       & opDat3Map, opDat3MapDim,  &
                       & 0, opSetCore%core_size, &
                       & numberOfOpDats,opArgArray,testfreq)

  ! wait for Halos to be received
  CALL op_mpi_wait_all(numberOfOpDats,opArgArray)

  ! compute over halo (redundant) iterations/elements
  CALL op_wrap_res_calc( opDat1Local, opDat3Local, &
  & opDat5Local, opDat7Local, &
  & opDat1Map, opDat1MapDim, opDat3Map, opDat3MapDim, &
  & opSetCore%core_size, n_upper,numberOfOpDats,&
  & opArgArray,2147483647)

  IF ((n_upper .EQ. 0) .OR. &
  &   (n_upper .EQ. opSetCore%core_size)) THEN
    CALL op_mpi_wait_all(numberOfOpDats,opArgArray)
  END IF

  ! mark halos dirty
  CALL op_mpi_set_dirtybit(numberOfOpDats,opArgArray)
  ....
  ....
END SUBROUTINE
END MODULE
```

https://github.com/OP-DSL/OP2-APPS/blob/main/apps/fortran/airfoil/airfoil_hdf5/res_calc_seqkernel.F90

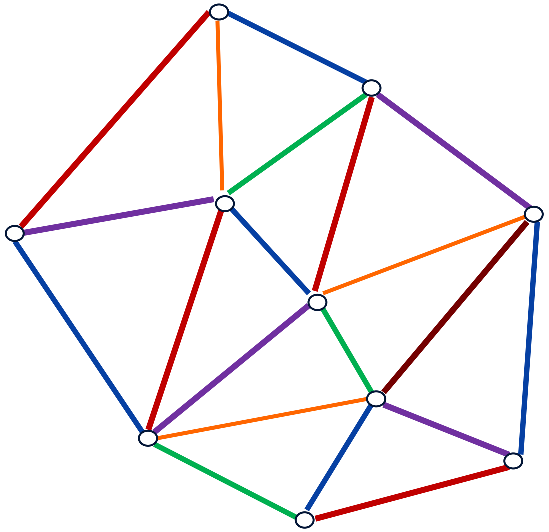
Handling Data-races

- Distributed memory parallelization
  - Mesh partitioning
  - Standard halo exchange methods
  - Redundant computation
- Single node – Inter-thread-block
  - Coloring
  - No two blocks of the same color update the same memory location
- Single node – Intra-thread block
  - Coloring
  - No two edges of the same colour update the same node
  - Use atomics





Threads 0 and 2 can run in parallel



OP2 – Generated Code – GPU with CUDA

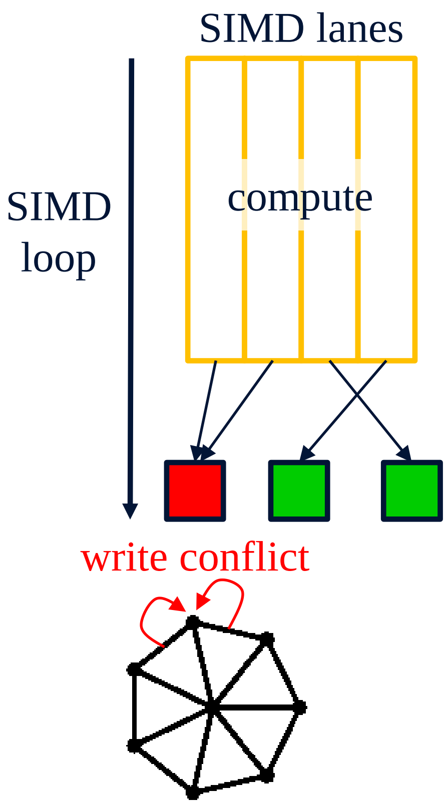
```fortran
SUBROUTINE res_calc_gpu(x1,x2,q1,q2,adt1,adt2,res1,res2)
  IMPLICIT NONE
  integer*4 istat
  REAL(kind=8)  :: x1(2)
  REAL(kind=8)  :: x2(2)
  REAL(kind=8), INTENT(IN) :: q1(4)
  REAL(kind=8), INTENT(IN) :: q2(4)
  REAL(kind=8), INTENT(IN) :: adt1
  REAL(kind=8), INTENT(IN) :: adt2
  REAL(kind=8) :: res1(4)
  REAL(kind=8) :: res2(4)
  REAL(kind=8) :: dx,dy,mu,ri,p1,vol1,p2,vol2,f
  dx = x1(1) - x2(1)
  dy = x1(2) - x2(2)
  ri = 1.0 / q1(1)
  p1 = 0.4 * (q1(4) - 0.5 * ri * (q1(2) * q1(2) + &
            & q1(3) * q1(3)))
  vol1 = ri * (q1(2) * dy - q1(3) * dx)
  ri = 1.0 / q2(1)
  p2 = 0.4 * (q2(4) - 0.5 * ri * (q2(2) * q2(2) + &
            & q2(3) * q2(3)))
  vol2 = ri * (q2(2) * dy - q2(3) * dx)
  mu = 0.5 * (adt1 + adt2) * 0.05
  f = 0.5 * (vol1 * q1(1) + vol2 * q2(1)) + &
           & mu * (q1(1) - q2(1))
  istat = atomicAdd(res1(1),+ f)
  istat = atomicAdd(res2(1),- f)
  f = 0.5 * (vol1 * q1(2) + p1 * dy + &
           & vol2 * q2(2) + p2 * dy) + mu * (q1(2) - q2(2))
  istat = atomicAdd(res1(2),+ f)
  istat = atomicAdd(res2(2),- f)
  f = 0.5 * (vol1 * q1(3) - p1 * dx + &
           & vol2 * q2(3) - p2 * dx) + mu * (q1(3) - q2(3))
  istat = atomicAdd(res1(3),+ f)
  istat = atomicAdd(res2(3),- f)
  f = 0.5 * (vol1 * (q1(4) + p1) + vol2 * (q2(4) + p2)) + &
           & mu * (q1(4) - q2(4))
  istat = atomicAdd(res1(4),+ f)
  istat = atomicAdd(res2(4),- f)
END SUBROUTINE
```

```fortran
! CUDA kernel function
attributes (global) SUBROUTINE op_cuda_res_calc( &
& opDat1Deviceres_calc, opDat3Deviceres_calc, &
& opDat5Deviceres_calc, opDat7Deviceres_calc, &
& opDat1Map, opDat3Map, start, end, setSize)
...
...
  i1 = threadIdx%x - 1 + (blockIdx%x - 1) * blockDim%x
  IF (i1+start<end) THEN
    i3 = i1+start
    map1idx = opDat1Map(1 + i3 + setSize * 0)
    map2idx = opDat1Map(1 + i3 + setSize * 1)
    map3idx = opDat3Map(1 + i3 + setSize * 0)
    map4idx = opDat3Map(1 + i3 + setSize * 1)
    ! kernel call
    CALL res_calc_gpu( &
    & opDat1Deviceres_calc(1+map1idx*(2):map1idx*(2)+2), &
    & opDat1Deviceres_calc(1+map2idx*(2):map2idx*(2)+2), &
    & opDat3Deviceres_calc(1+map3idx*(4):map3idx*(4)+4), &
    & opDat3Deviceres_calc(1+map4idx*(4):map4idx*(4)+4), &
    & opDat5Deviceres_calc(1+map3idx), &
    & opDat5Deviceres_calc(1+map4idx), &
    & opDat7Deviceres_calc(1+map3idx*(4):map3idx*(4)+4), &
    & opDat7Deviceres_calc(1+map4idx*(4):map4idx*(4)+4))
  END IF
END SUBROUTINE
```

https://github.com/OP-DSL/OP2-APPS/blob/main/apps/fortran/airfoil/airfoil_hdf5/res_calc_kernel.CUF

PARALLELIZING ON MULTI-CORE CPUS : SIMD VECTORIZATION – THE PROBLEM

- Aim – execute computation on multiple edges simultaneously
- For DP mathematics, multiple = 4 (256 bits vector length) or 8 (512 bits vector length)



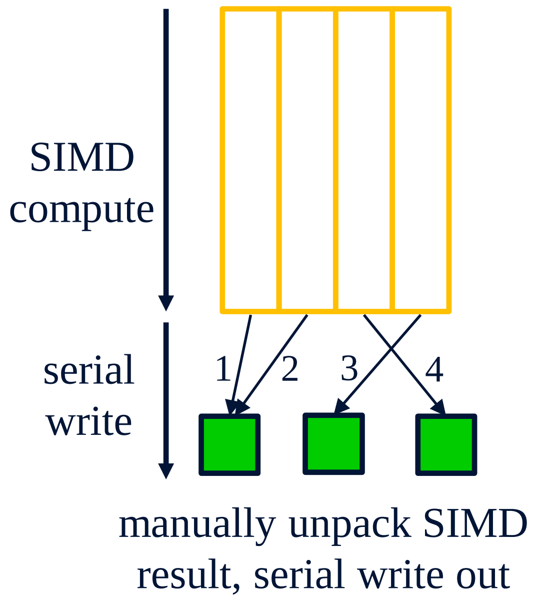
Incorrect solution!

Technique : Gather / Scatters

- Gather edge data to vector length local arrays
- Pass local arrays as arguments to kernel accepting “vectorized” arguments
- Apply nodal update as serial loop !

Issues

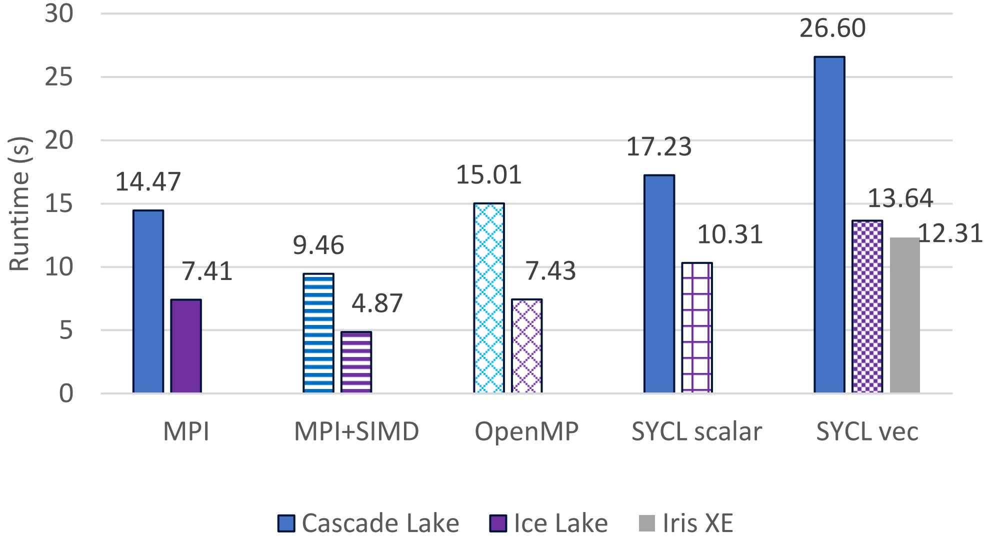
- Need new kernel that accepts vectorized arguments
- Extra overhead due to gather/ scatters
- Not all kernels will benefit from vectorization
- Best for only highly computationally intensive kernels

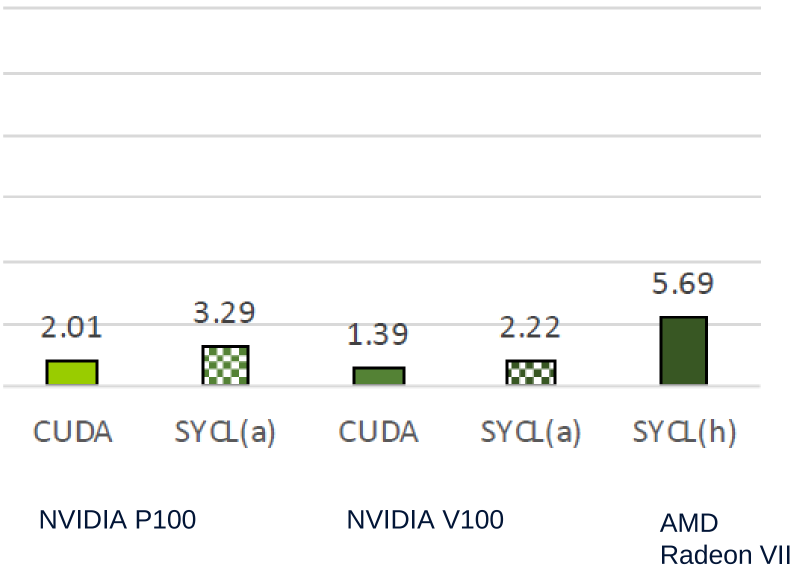


https://github.com/OP-DSL/OP2-APPS/blob/main/apps/c/airfoil/airfoil_hdf5/dp/vec/res_calc_veckernel.cpp

OP2 –CPU VS GPU WITH SYCL (COLORING VS ATOMICS)

- MG-CFD – Multigrid CFG MiniAPP:
  - NASA Rotor37, 4 multigrid levels, 8M edges
  - Generate Parallelization using OP2
  - Intel compilers - from oneAPI
  - Intel MPI - for MPI, SIMD, OpenMP, MPI+OpenMP
- GPUs – NVIDIA P100 and V100, AMS Radion VII, Intel Iris XE MAX
- CPUs – single socket only to avoid NUMA issues:
  - Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz, 16 cores
  - Intel(R) Xeon(R) Platinum 8360Y @ 2.40 GHz, 36 cores
  - SYCL compilers - Intel OneAPI 2021.4 and HipSYCL



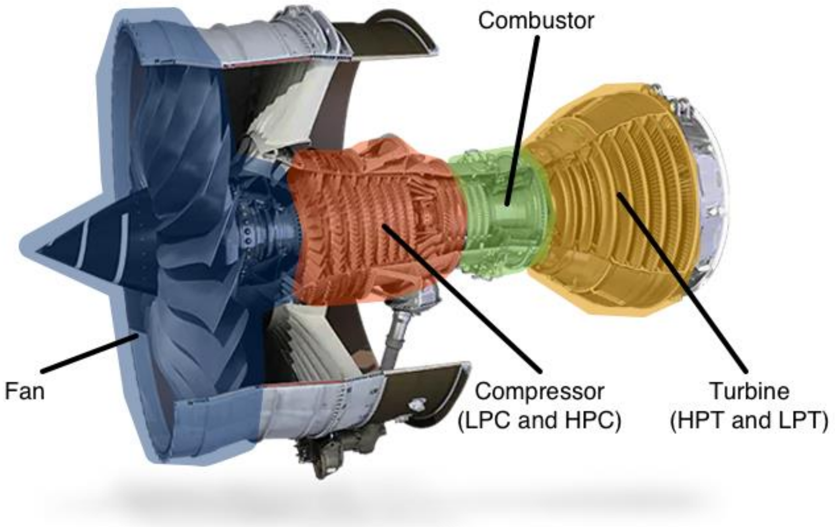


I.Z. Reguly, A.M.B. Owenson, A. Powell, S.A. Jarvis, and G.R. Mudalige, *Under the Hood of SYCL – An Initial Performance Analysis With an Unstructured-mesh CFD Application*, International Supercomputing Conference (ISC 2021), June 2021.

I.Z. Reguly. *Performance of DPC++ on Representative Structured/Unstructured Mesh Applications.* Intel DevSummit at SC21

- Virtual certification of Gas Turbine Engines – EPSRC Prosperity Partnership (ASIMOV)
  - Main consortium with partners – EPCC, Warwick, Oxford, Cambridge, Bristol and Rolls-Royce plc.

- Grand Challenge 1 – Sliding Planes model of Rig250 (DLR test rig compressor)
  - 4.5 stage rotor-stator (10-row full annulus) | 4.58B mesh nodes.
  - Need to obtain 1 revolution of compressor in less than 24 hours
  - Current production estimates at 7 days

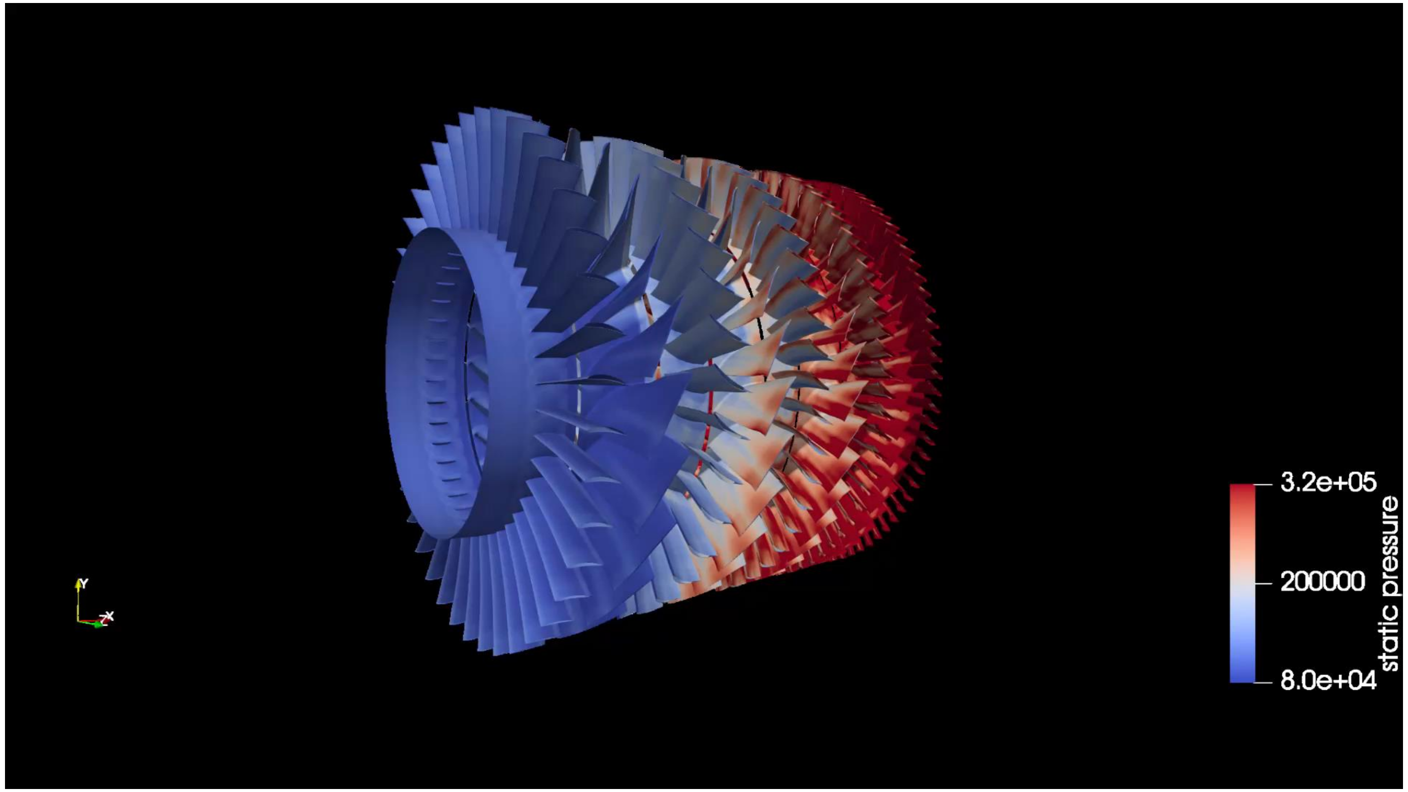


- Setup
  - Moving rotor-stator – sliding planes interfaces
  - Rotors and Stators modelled with Hydra CFD suite – URANS (360 degree models)
  - 10 rotor-stator interfaces
  - Code coupling for sliding planes – move from current monolithic (Hydra only) production code to coupling

- Challenges
  - Performance portability – run both CPUs and GPUs by multiple vendors
  - Preserve production code's scientific code and structure – cannot re-write, MUST "evolve" not overhaul !
  - Convince users to adopt ! (Ongoing for nearly 10 years now)

OP2-Hydra Performance *

System	ARCHER2 HPE Cray EX [6]	Cirrus SGI/HPE 8600 GPU Cluster [4]
Processor	AMD EPYC 7742 @ 2.25 GHz	Intel Xeon Gold 6248 (Cascade Lake) @ 2.5 GHz + NVIDIA Tesla V100-SXM2-16GN GPU
(procs×cores) /node	2×64	2×20 + 4×GPUs
Memory/node	256 GB	384 GB + 40GB/GPU
Interconnect	HPE Cray Slingshot 2×100 Gb/s bi-directional/node	Infiniband FDR, 54.5 Gb/s
OS	HPE Cray LE (based on SLES 15)	Linux CentOS 7
Compilers	GNU 10.2.0	nvfortran (nvhpc 21.2)
Compiler Flags	`-O2 -eF -fPIC`	
Power/node	660W	≈ 900W



OP2-Hydra Performance *

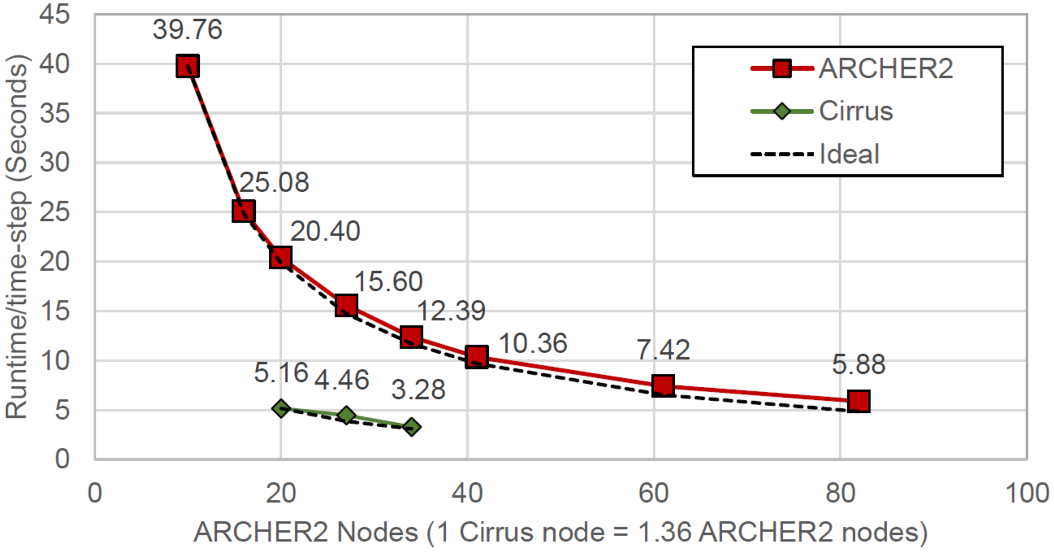
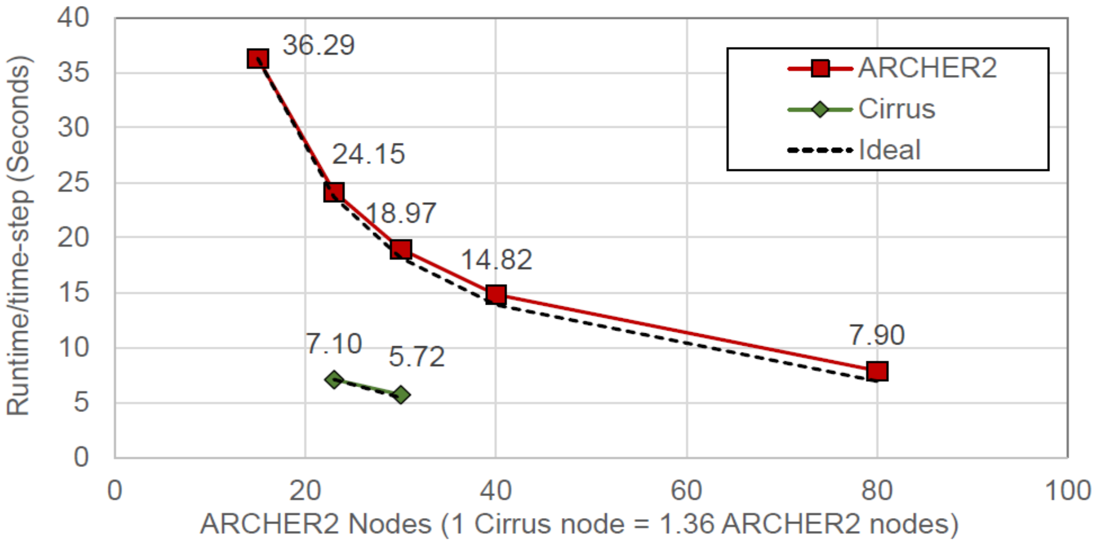


Figure 7: Rig250 $1 - 10_{430M}$ Mesh Runtime



Rig250 $1 - 2_{653M}$ Mesh Runtime

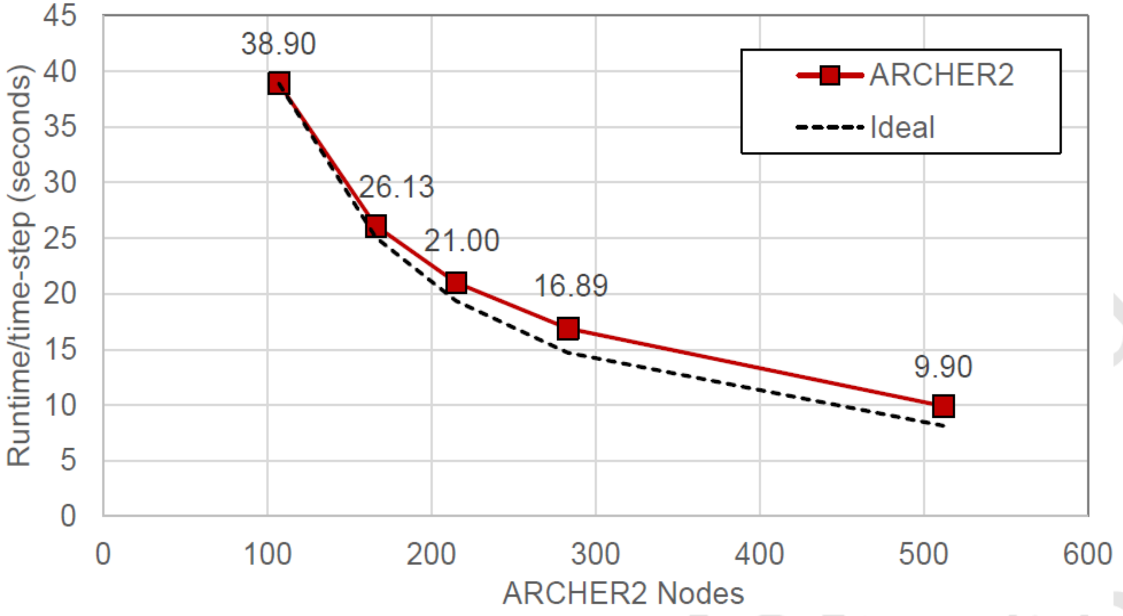
- ❑ ARCHER2 @ 34 nodes
  - 94% parallel efficiency
  - 10% coupling overhead
- ❑ Cirrus @ 25 nodes
  - 94% parallel efficiency
  - 20% coupling overhead
- ❑ ARCHER2 @ 82 nodes
  - 82% parallel efficiency
  - 20% coupling overhead

3.7- 4x speedup

- ❑ ARCHER2 @ 80 nodes
  - 88% parallel efficiency
  - 8% coupling overhead
- ❑ Cirrus @ 22 nodes
  - 94% parallel efficiency
  - 12% coupling overhead
- ❑ Cirrus @ 22 nodes
  - 94% parallel efficiency
  - 12% coupling overhead

3.3 - 3.4x speedup

* Results under review



Rig250 $1-10_{4.58B}$ Mesh Runtime

Achieved (A) and *Projected (P)* times to solution (hours) : Rig250, 1 revolution

Rig250 Problem	ARCHER2		Cirrus	
	Runime	#nodes	Runtime	#nodes
$1-10_{430M}$ - Monolithic	*93.0 (P)*	*8*		
$1-10_{430M}$ - Coupled	*85.0 (P)*	*8*	*2.9 (P)*	*15*
$1-10_{430M}$ - Coupled	*3.3 (P)*	*80*	*1.8 (P)*	*25*
$1-2_{653M}$ - Monolithic	*110.0 (P)*	*8*		
$1-2_{653M}$ - Coupled	*40.0 (P)*	*8*	*3.9 (P)*	*17*
$1-2_{653M}$ - Coupled	*8.2 (P)*	*40*	*3.2 (P)*	*22*
$1-10_{4.58B}$ - Coupled	*14.5 (A)*	*166*	*4.7 (P)*	*122*
$1-10_{4.58B}$ - Coupled	*9.4 (A)*	*256*		
$1-10_{4.58B}$ - Coupled	*5.5 (A)*	*512*		

- ARCHER2 @ 512 nodes:
  - 82% parallel efficiency (vs 107 node run)
  - 15% coupling overhead

- 122 Cirrus nodes is power equivalent to 166 ARCHER2 nodes
- ARCHER2 needs just over 3x more number of *power equivalent* nodes (512) to match Cirrus's runtime (4.7 hours)

* Results under review

CROSS-LOOP TECHNIQUES

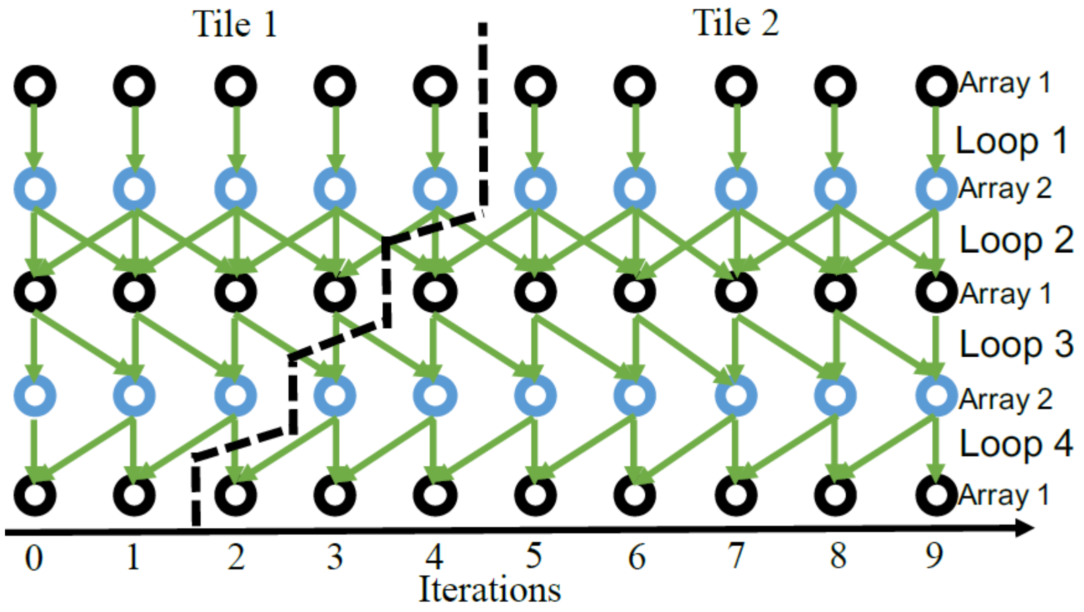
- Loop descriptors and user contract allows to delay the execution of loops until API call to return data to user

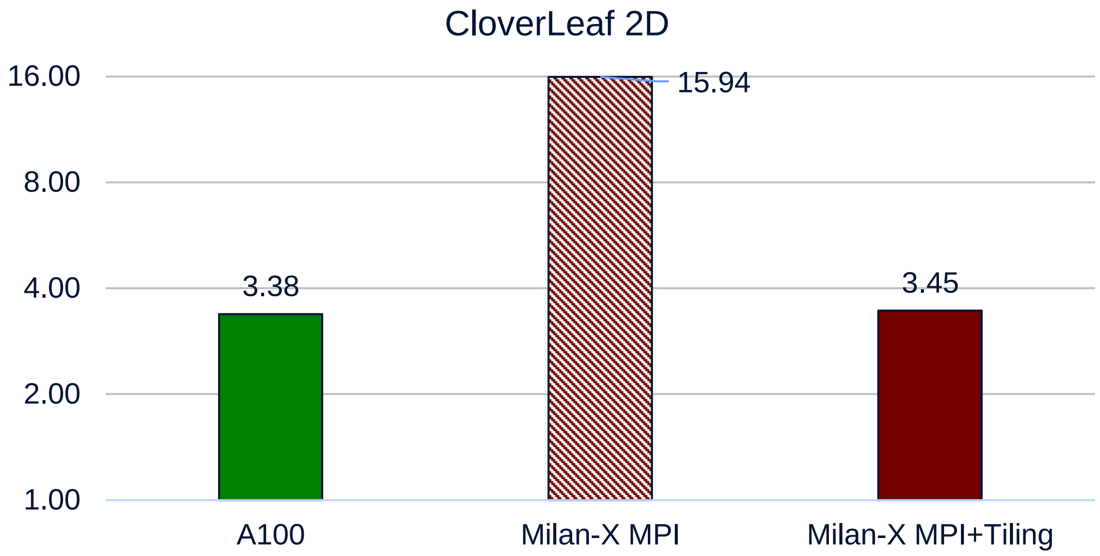
- Now we have information about a sequence of loops to analyze/reason about together
  - Access descriptors provide precise dependence iteration-to-iteration information
  - Reason about a chain (DAG) of parallel loops at runtime

- Cross-loop optimizations
  - Cache-blocking Tiling
  - Distributed memory communication avoidance
  - Automated checkpointing – only checkpoint the absolutely necessary data

- No changes to the high-level user code

Cache-blocking Tiling





- ❑ Data sets too large to fit on cache : limited data reuse
- ❑ Improve reuse by considering multiple loops
- ❑ Need to make sure all data dependencies are satisfied
  - Block iteration ranges of loops,
  - reorganize them so that data accessed by a given block in the first loop nest stays in cache and gets accessed by blocks of subsequent loop nests
  - Parallelize within tiles

AMD Milan-X (Azure HBv3) vs A100
4TB/s L3 cache BW
* Recent runs done by Istvan Reguly PPCU.

- ❑ Tiling done over many loops spread across many compilation units
- ❑ Many complex loops
- ❑ Can't be done by existing (compiler) technology

PROJECTS AND CODES USING OP-DSLS - OPENSBLI

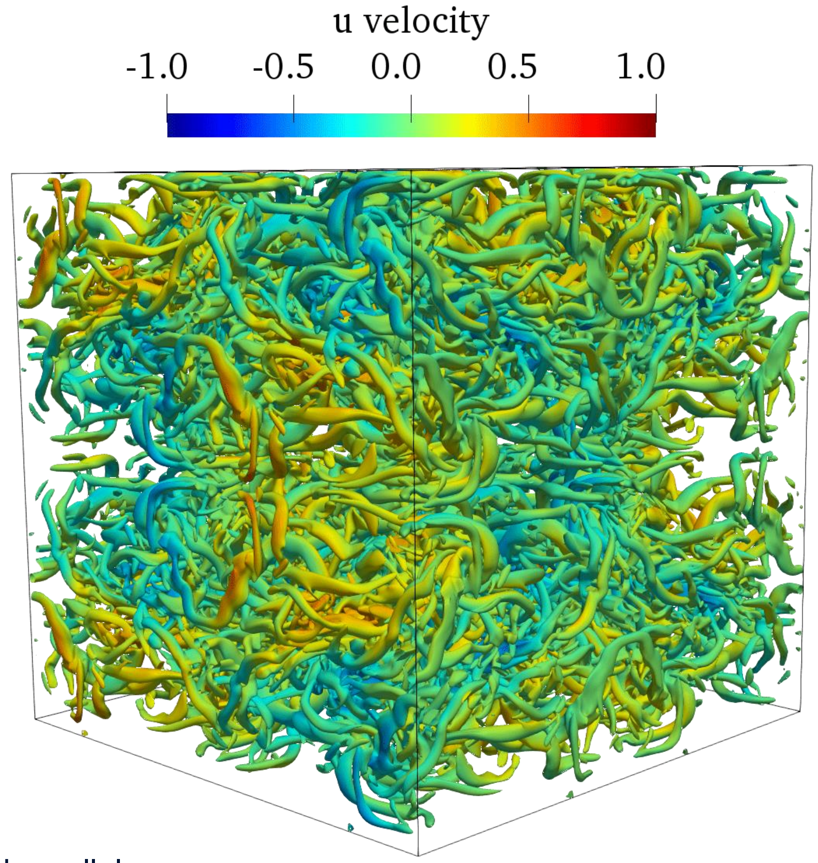
- Compressible Navier-Stokes solver
  - With shock capturing WENO/TENO
  - 4th order Finite Difference
  - Single/double precision

- OpenSBLI is a Python framework
  - Write equations in SymPy expressions
  - OPS code generated

```
ndim = 3
sc1 = "**{\'scheme\':\'Teno\'}"
# Define the compresible Navier-Stokes equations in Einstein notation.
mass = "Eq(Der(rho,t), - Conservative(rhou_j,x_j,%s))" % sc1
momentum = "Eq(Der(rhou_i,t) , -Conservative(rhou_i*u_j + KD(_i,_j)*p,x_j , %s) + Der(tau_i_j,x_j) )" % sc1
energy = "Eq(Der(rhoE,t), - Conservative((p+rhoE)*u_j,x_j, %s) - Der(q_j,x_j) + Der(u_i*tau_i_j ,x_j) )" % sc1
stress_tensor = "Eq(tau_i_j, (mu/Re)*(Der(u_i,x_j)+ Der(u_j,x_i) - (2/3)* KD(_i,_j)* Der(u_k,x_k)))"
heat_flux = "Eq(q_j, (-mu/((gama-1)*Minf*Minf*Pr*Re))*Der(T,x_j))"
# Numerical scheme selection
Avg = RoeAverage([0, 1])
LLF = LLFTeno(teno_order, averaging=Avg)
cent = Central(4)
rk = RungeKuttaLS(3, formulation='SSP')
# Specifying boundary conditions
boundaries[direction][side] = IsothermalWallBC(direction, 0, wall_eqns)
# Generate a C code
alg = TraditionalAlgorithmRK(block)
OPSC(alg)
```

OpenSBLI

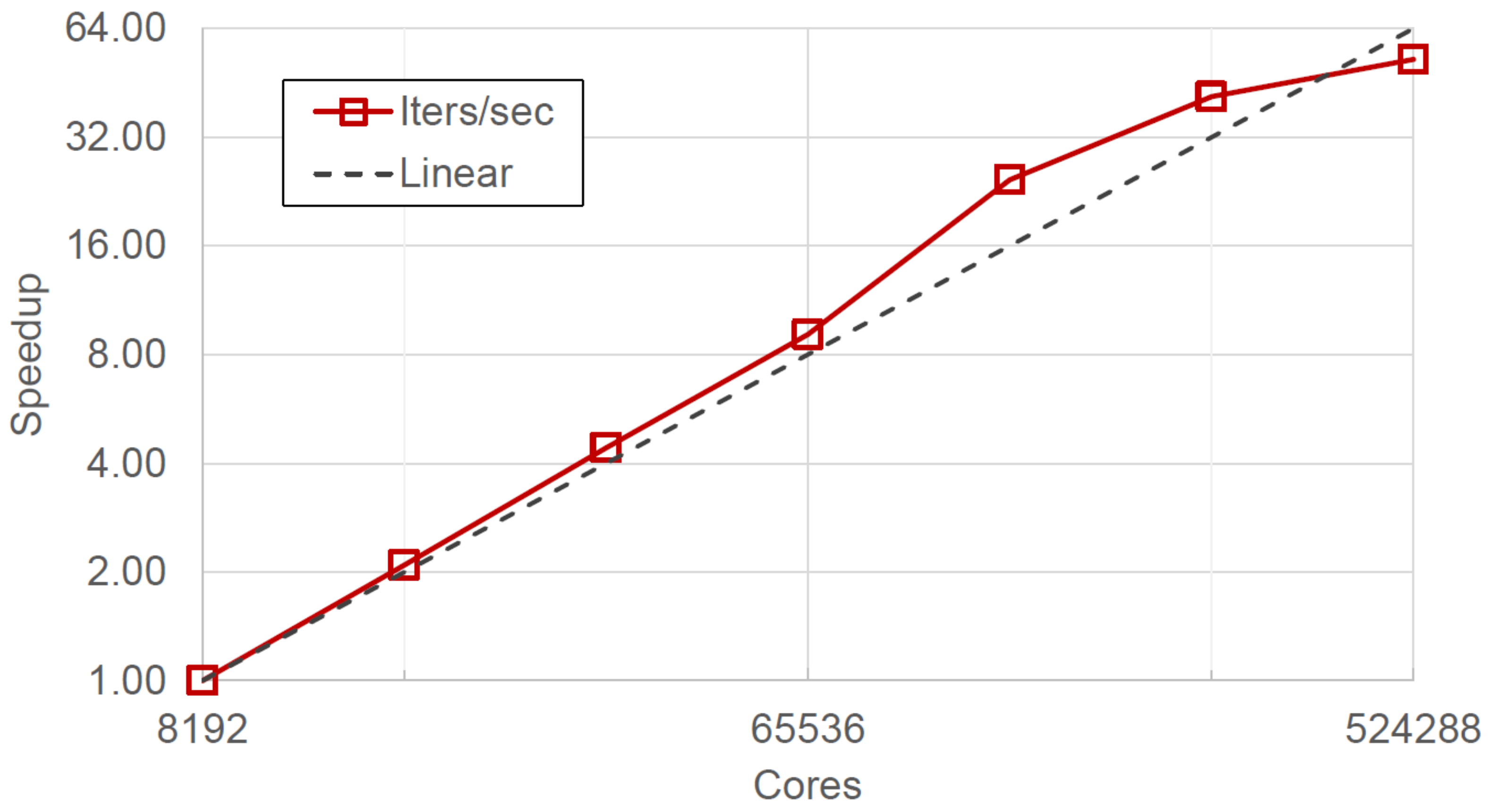
https://opensbli.github.io/



Jacobs, C. T., Jammy, S. P., Sandham N. D. (2017). OpenSBLI: A framework for the automated derivation and parallel execution of finite difference solvers on a range of computer architectures. Journal of Computational Science, 18:12-23, DOI: 10.1016/j.jocs.2016.11.001

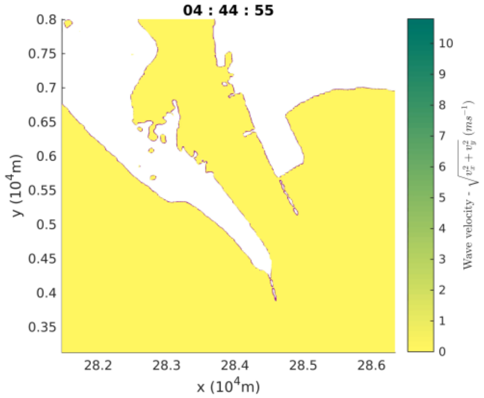
- Taylor – Green Vortex Problem – ARCHER2 benchmark
  - Strong Scaling - 1024^3 Mesh
  - Double precision
  - 128 MPI processes per node
  - Speedup calculated from 1000 iterations – includes start up time.

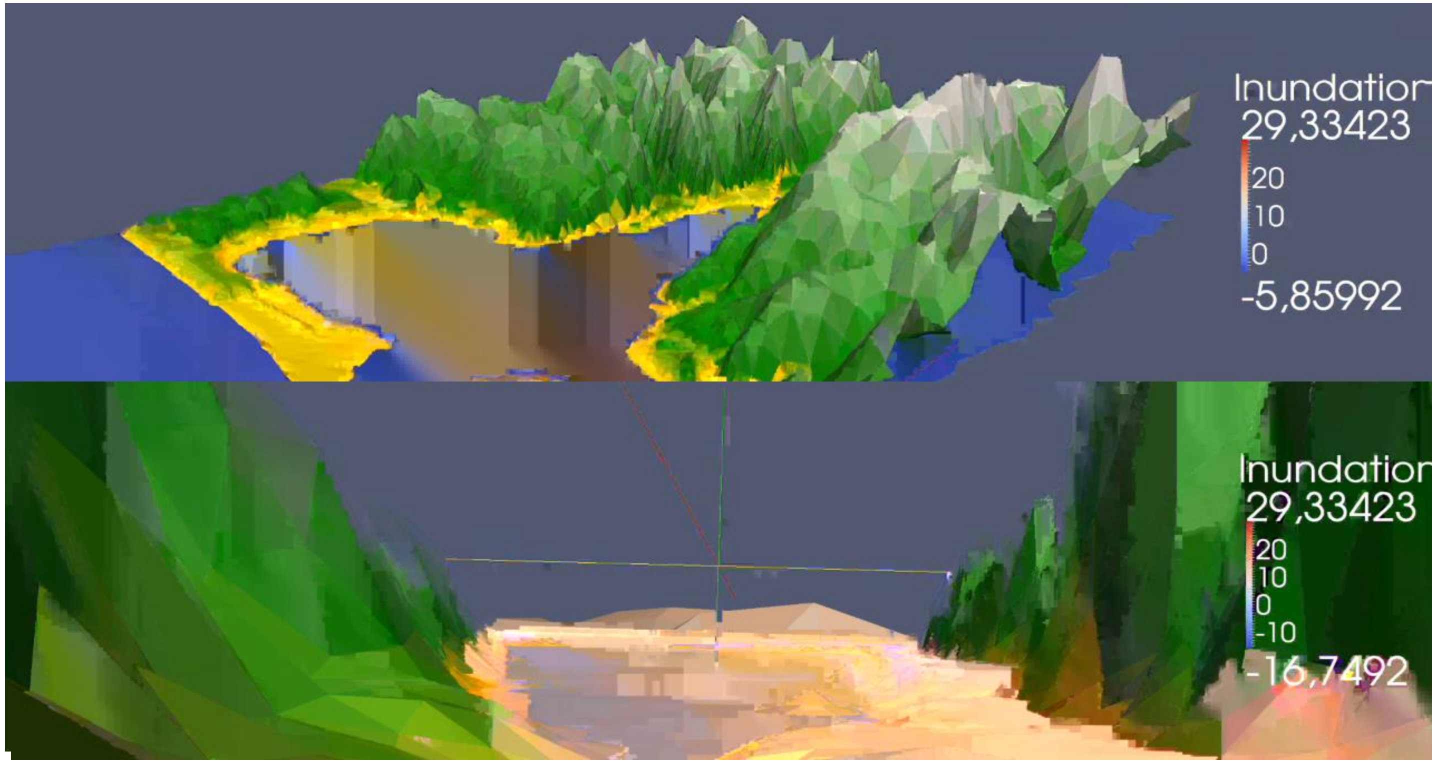
From recent benchmarking runs done by Andrew Turner and the ExCALIBUR Benchmarking team (Oct 2021)

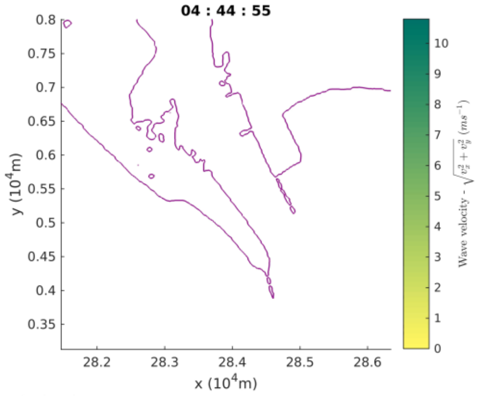


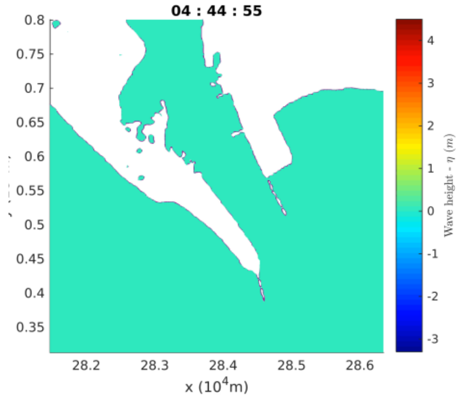
- Work with UCL and ATI
  - Develop tsunami emulators with UQ
  - Validate with simulation
  - Almost real-time simulation on GPUs

https://github.com/reguly/volna









Reguly, I. Z., Giles, D., Gopinathan, D., Quivy, L., Beck, J. H., Giles, M. B., Guillas, S., and Dias, F.: The VOLNA-OP2 tsunami code (version 1.5), Geosci. Model Dev., 11, 4621–4635, https://doi.org/10.5194/gmd-11-4621-2018, 2018.

Challenges – Cost / Effort of Conversion

- Converting legacy code is time consuming
  - Large code base
  - Defunct 3rd party libs
  - Fortran 77 or older !
- Difficult to validate code
  - New code giving the same accurate scientific output ?
  - What code should I certify ? High-level code/generated code ?
  - Difficult to convince users to use new code - fear of an opaque compiler / intermediate representation / black box !
- Incremental conversion – loop by loop
  - Simpler than CUDA, but more difficult than OpenACC/OpenMP
  - Automated conversion ?
- Changing user requirements
  - Wanting to use a DSL for doing things beyond what it was intended for !
  - Asking for “back-doors” / “escape hatches” -- leads to poor performance

Challenges – Cost / Effort of Conversion

- Tools not entirely mature
  - Currently source-to-source with Python
  - Pushing clang/LLVM source-to-source to do what we want
  - What about Fortran - may be F18/Flang ?
  - MLIR appearing to give some advance capabilities – see ExCALIBUR xDSL project (Tobias Grosser, Paul Kelly et al.)
- Code-generation for more exotic architectures – e.g. FPGAs
  - Large design space
  - Complex source transformations –cross loop, loop fusion and unrolling to create longer and longer pipelines !
- Maintainable/long term source-to-source technologies
  - Domain Scientists not having expertise to understand / maintain DSLs

Challenges – Cost / Effort of Conversion

- Currently purely done via academic and (small/short term) industrial funding

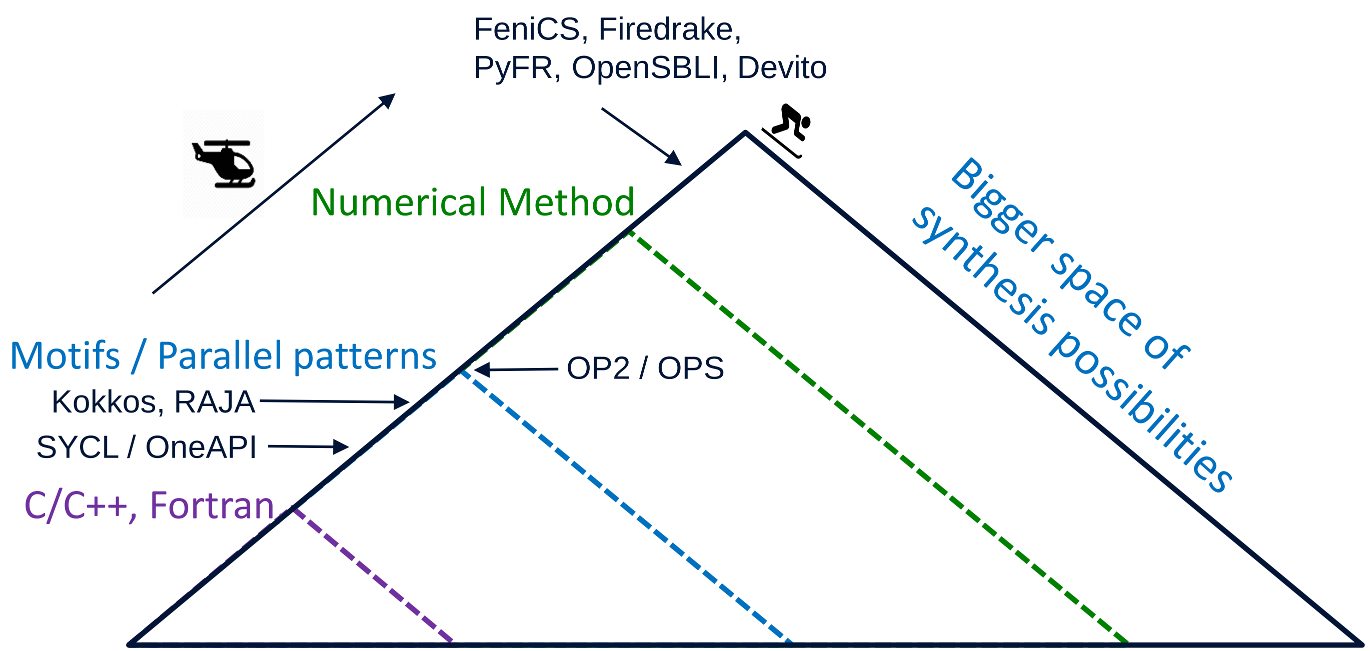
- Long term funding and maintenance
  - Once established probably will not be different to any other classical library
  - Will require compiler expertise to maintain code generation tools

- What DSL to choose ?
  - Re-use technologies / DSLs – especially code-gen tools (best not to reinvent !)

- Skills Gap
  - Programme in C/C++/Fortran (at a minimum)
  - Knowledge of compilers / code-generation
  - Compete for applicants – Communicate what we do better | impact of HPC / Computational Sciences
  - Salary
  - Contracts

DSLs / High-level Abstractions gaining traction

- ❑ FEniCS - PDE solver package - https://fenicsproject.org/
- ❑ Firedrake - automated system for the portable solution of PDEs using the finite element method https://www.firedrakeproject.org/
- ❑ PyFR - Python based framework for solving advection-diffusion type problems on streaming architectures using the Flux Reconstruction approach - http://www.pyfr.org/
- ❑ Devito - prototype DSL and code generation framework based on SymPy for the design of highly optimised finite difference kernels for use in inversion methods - http://www.opesci.org/devito-public
- ❑ GungHO project - Weather modelling codes (MetOffice)
- ❑ STELLA – DSL for stencil codes(Metro Swiss)
- ❑ Liszt – Stanford University : DSL for solving mesh-based PDEs - http://graphics.stanford.edu/hackliszt/
- ❑ Kokkos – C++ template library – SNL
- ❑ RAJA - C++ template libraries - LLNL



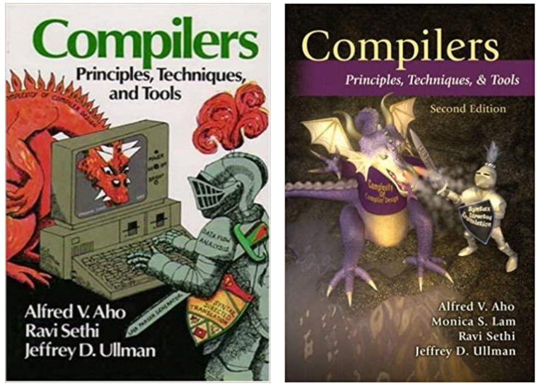
Adapted from: *Synthesis versus Analysis: What Do We Actually Gain from Domain-Specificity?* Keynote talk at the LCPC 2015. Paul H. J. Kelly (Imperial College London)

ONGOING WORK AND FUTURE PLANS

- CCP-Tubulance
  - Direct solver libraries – Tri-, penta-, 7-, 9-, 11 diagonal, multi-dimensional solvers
  - Integrate directsolver libraries to be called within OPS
  - OpenSBLI type high-level (Python) framework for XCompact3D – High Order FD framework
- ExCALIBUR Phase 1B – Turbulence at the Exascale
  - Imperial, Warwick, Newcastle, Southampton, Cambridge, STFC collaboration | UKTC and UKCTRF Communities
  - Xcompact3D and Wind Energy, OpenSBLI and Green Aviation, uDALES and Air Quality, SENGA+ and Net-Zero Combustion
  - Extending OPS capability – robust code-gen tools and parallel transformations | support future-proof code development
  - UQ, I/O, Coupling and Visualization
  - Machine Learning Algorithms for Turbulent Flow
- Task-based parallelism (Legion from Stanford)
- In-situ visualization
- AMR – some on-going work, but difficult to get a good abstraction

Conclusions

- Utilizing domain knowledge will expose things that the compiler does not know
  - Iterating over the same mesh many times without change
  - Mesh is partitioned and colorable
- Compilers are conservative
  - Force it to do what you know is right for your code !
- Let go of the conventional wisdom that higher abstraction will not deliver higher performance
  - Higher abstraction leads to a bigger space of code synthesis possibilities
  - We can automatically generate significantly better code than what (most) people can (reasonably) write
  - Do not destroy performance portability by (hand-) tuning at a very low level to a specific platform



"Fundamentals and abstractions have more staying power than the technology of the moment" Alfred Aho and Jeffrey Ullman
(Turing Award Recipients 2020)

ACKNOWLEDGEMENTS

- OP2 was part-funded by the UK Technology Strategy Board and Rolls-Royce plc. through the SILOET project, and the UK EPSRC projects EP/I006079/1, EP/I00677X/1 on Multi-layered Abstractions for PDEs.

- OPS was part-funded by the UK Engineering and Physical Sciences Research Council projects EP/K038494/1, EP/K038486/1, EP/K038451/1 and EP/K038567/1 on "Future-proof massively-parallel execution of multi-block applications" and EP/J010553/1 "Software for Emerging Architectures" (ASEArch) project.

- Rolls-Royce plc., and by the UK EPSRC (EP/S005072/1) Strategic Partnership in ComputationalScience for Advanced Simulation and Modelling of Engineering Systems (ASiMoV).
- OpenSBLI was part-funded by EPSRC grants EP/K038567/1 and EP/L000261/1, and European Commission H2020 grant 671571 "ExaFLOW: Enabling Exascale Fluid Dynamics Simulations

- Gihan Mudalige was supported by the Royal Society Industrial Fellowship Scheme (INF/R1/180012)
- Istvan Reguly was supported by the Janos Bolyai Research Scholarship of the Hungarian Academy of Sciences.

- Thematic Research Cooperation Establishing Innovative Informatic and Info-communication Solutions, which has been supported by the European Union and co-financed by the European Social Fund under grant number EFOP-3.6.2-16-2017-00013.

- UK National Supercomputing Service – ARCHER2 and resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

Downloads and More Information

- GitHub Repositories
  - OP2 – https://github.com/OP-DSL/OP2-Common
  - OPS – https://github.com/OP-DSL/OPS

- OP-DSL Webpage - https://op-dsl.github.io/

- Contact
  - Gihan Mudalige (Warwick) - g.mudalige@warwick.ac.uk
  - Istvan Reguly (PPCU – Hungary) - reguly.istvan@itk.ppke.hu